Read-type Routines

Answers from internal ptrs

Oct 12, 1990

A data request made to a local station is "compiled" into an array of internal ptrs. These ptrs are interpreted by "read-type" routines according to the listype(s) used in the request. The read-type routine is called to process an array of internal ptrs that correspond to the array of idents in the original data request and produce the corresponding answers. Because an *array* of internal ptrs is passed to the routine, it is optimized in speed; its logic does the same processing for each element of the array.

As new listypes are added to the system, the support often requires new read-type routines. In the spirit of the named downloaded programs that are used with local applications, this note considers the implementation of named downloaded read-type routines.

Current branch table

The current scheme for selecting the read-type routine involves a table of two-byte offsets to the code (to provide for position independence) for each read-type routine. The read-type# from the listype table is the table index.

As the system grows—the current size is 53K—the use of two-byte offsets may appear limiting, so 4-byte offsets could be used. System changes currently underway will concentrate the knowledge of this branch table into a single routine called READTYPE, whose single argument is the read-type#, so such a change would be simple.

New selection scheme

The new branch table has 4-byte offsets for each resident routine but 4-character names for each downloadable routine. The read-type branch table is scanned. Each offset entry is converted to a ptr to the resident code, and each name entry is converted to a ptr to the executable copy using entries (of type RTYP) found in the CODES table. (For each RTYP entry found, the download copy is sum-checked and copied into a newly-allocated "executable" memory block.) The ptr to the read-type routine is stored in a ram-based table used by READTYPE. There is no provision for replacing or adding a read-type routine without going through the initialization logic at reset time.